TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikka, ohjelmistotekniikka

Tutkintotyö

Rikhard Nousiainen

OHJELMISTOPAKETTIEN KOKOAMISEN HALLINNOINTI JA AUTOMATISOINTI

Työn valvoja: Lehtori Erkki Hietalahti
Työn teettäjä: SYSOPENDIGIA Oyj, engineering manager Pasi Pitkänen
Tampere 2007

Nousiainen, Rikhard
Tutkintotyö
Työn valvoja
Työn teettäjä
Huhtikuu 2007
Hakusanat

Ohjelmistopakettien kokoamisen hallinnointi ja automatisointi
I – VIII, 31 sivua
Lehtori Erkki Hietalahti, build engineer Kari Lajunen
SYSOPENDIGIA Oyj, engineering manager Pasi Pitkänen

Ohjelmistopaketti, ohjelmistopakettien kokoaminen, konfiguraationhallinta

TIIVISTELMÄ

Tämän insinöörityön tarkoituksena on selventää, millainen työnkuva ohjelmistopakettien kokoajalla on, millaisia työkaluja hän joutuu työssään käyttämään ja miten hänen työtään voisi helpottaa automatisoinnin avulla. Työn yksi tärkeimmistä tehtävistä on valottaa konfiguraationhallinnan toiminnallisuutta ohjelmistoteollisuudessa. Konfiguraationhallintaa käytetään yleisesti laajojen ohjelmistokokonaisuuksien rakentamisen hallinnoimiseen. Ohjelmistopakettien kokoaja toimii konfiguraationhallinnan sääntöjen alaisena ja käyttää monipuolisia työkaluja, jotta pakettien kokoaminen olisi mahdollisimman hallittua työtä.

Konfiguraationhallinnassa panostetaan suunnitteluun sekä menetelmien ja prosessien määrittelyyn. Kun tämä vaihe on kehittynyt tarpeeksi korkealle tasolle, voidaan manuaalista työtä vähentää automatisoimalla konfiguraationhallinnassa käytettävien työkalujen toiminnallisuuksia.

Ohjelmistopakettien kokoaja tekee projektin loppuvaiheessa julkaisun, joka toimitetaan eteenpäin, joko asiakkaalle, firman sisäiseen tarkoitukseen tai maailmalle. Kokoajan tärkein työkalu on versionhallintatyökalu, jota käytetään jokaisen ohjelmistoprojektin eri työvaiheiden kartoittamiseen ja erilaisten tiedostojen, komponenttien ja materiaalien säilytyspaikkana. Automatisoinnin avulla ohjelmistopakettien kokoajan työtä voidaan helpottaa ja nopeuttaa.

Nousiainen, Rikhard Build Management and automation of software package
Engineering Thesis I – VIII, 31 pages
Thesis Supervisor lector Erkki Hietalahti
Commissioning Company SYSOPENDIGIA Plc, engineering manager Pasi Pitkänen
April 2007
Keywords Software package, Build Management, Configuration management, Software building management, Build engineer

ABSTRACT

Purpose of this thesis work was to clear out and analyze what kind of work does build engineer do when he builds software packages and what kind of tools does he has to use at his work. One of the most important subjects of this work is to describe the functionality of software configuration management. Configuration management is being used widely for managing software projects. Build engineer follows the rules of software configuration management and uses proper tools so software packaging would be as well managed as possible.

Configuration management invests on designations, procedures and process definitions. When this phase has reached level high enough, manual work can be reduced by using automation at the functionality in the tools used at configuration management.

Build engineer makes a software release at the end of each version in projects and delivers the package for customer, internal use or publishes it for the world to use. Build engineers most important tool to work with is version control tool what is being used for every phase in software projects and to store all kinds of material in it. Automation enables faster building for build engineers and also eases workload.

ALKUSANAT

Tämä tutkintotyö on tehty SYSOPENDIGIA Oyj:lle 2007. Mielenkiintoni ohjelmistopakettien kokoamiseen tulee omasta työkokemuksestani ohjelmistopakettien kokoajana. Ammattikorkeakoulussa tutustuin yleisesti ohjelmistopakettien kehittämiseen ja ohjelmien kääntämiseen mm. Unix-ympäristössä toimivalla GCC-kääntäjällä. Tämä kääntäjä on täysin komentorivipohjainen ja sisältää erittäin paljon toiminnallisuutta. Aloittaessani nykyisessä työssäni SYSOPENDIGIA Oyj:ssä konfiguraationhallinan puolella huomasin, kuinka paljon kyseistä kääntäjää käytetään erilaisten ohjelmien kokoonpanossa. Tästä sainkin idean selvittää, miten ohjelmistopakettien kokoaminen tehdään hallitusti ja mahdollisimman automatisoidusti. Selvityksen suurin työ oli kerätä tarpeeksi kokemusta ja aineistoa, jota apuna käyttäen pystyy kertomaan suurien ohjelmistoprojektien ohjelmistopakettien kokoamisesta ja peilaamaan käytettävissä olevaa aineistoa omaan työkokemukseeni vastaavassa työssä.

Haluaisin kiittää kaikkia, jotka ovat auttaneet tämän tutkintotyön tekemisessä.

Tampereella 23. huhtikuuta 2007

_____

Rikhard Nousiainen

SISÄLLYSLUETTELO

KUVAT

KÄYTETYT ERIKOISTERMIT

Konfiguraatio	Mahdollistaa ohjelmiston kehityksen seurannan ja hallitun kehityksen
Identifionti	Konfiguraationhallinnan osa-alue, jossa yksilöidään sovellus
Bugi	Ohjelmiston lähdekoodissa esiintyvä virhe
Perustuote	Luodaan versionhallinnassa, jotta saadaan varmistettua ohjelmiston yhtenäisyys (engl. baseline product)
Kantaversio	Jäädytetyssä tilassa oleva pohja kehittäjien avuksi, sekä projektin etenemisen merkkipaalu
Muutosvaatimus	Ohjelmistossa syntyy muutoksia (mm. speksit), jotka täytyy saada tehtyä ohjelman lähdekoodiin
Auditointi	Ulkopuolinen taho suorittaa laaduntarkistuksen ja mahdollisten virheiden esiintymistä
V&V	Validointi & Verifiointi- tekniikka, jossa tutkitaan tuotteen oikeanlaista rakentamista
SQA	Ohjelman tila on varmennettu toimivaksi (engl. Software Quality Assured)
Skripti	Hoitaa automatisoinnissa mekaanisen tekemisen

Spesifikaatio	Dokumentti, jossa on tiedot tuotteesta tai palvelusta
Käänteistekniikka	Käytetään tietyn valmiin tuotteen jäljittämisessä (engl. reverse engineering)
Filtteröinti	Lajitellaan pois tiettyjä asioita, joiden ei tahdota näkyvän
Binäärikoodi	Vain tietokoneen ymmärtämä muoto
Replikointi	Tiedonvaihtoa ja keskustelua tietokantojen välillä

1 JOHDANTO

Ohjelmistotekniikka on kehittynyt suuresti muutamassa vuosikymmenessä. Yhden ihmisen ohjelmistoprojektit ovat jo suurimmaksi osaksi historiaa ja nykyisin ohjelmistoja kehitetään projektien koon mukaan erisuuruisissa ryhmissä. Ohjelmistojen tekeminen painottuu enemmän laadukkuuteen, varmuuteen ja nopeuteen kuin aikaisempina vuosina ja ohjelmistoprojektit pyritään yleensä jakamaan mahdollisimman sopiviin osiin. Ohjelmistojen konfiguraationhallinnalla voidaan kontrolloida ja hallita ohjelmistojen rakentamista. Konfiguraationhallinta koostuu eri osa-alueista pitäen sisällään mm. identifiointia, versionhallintaa ja keinon hallita kaikenlaisia muutoksia ohjelmistoprojekteissa.

Ohjelmistokehityksen keskeisiä työntekijöitä konfiguraationhallinnassa ovat ohjelmistopakettien kokoaja ja ohjelmistopakettien julkaisija. Tämän tutkintotyön tavoitteena on antaa mahdollisimman kattava kuvaus ohjelmistopakettien kokoajan eri työvaiheista ja yleistä tietoa siitä, millaisia työkaluja pakettien kokoaminen ja rakentaminen vaatii. Tämän lisäksi tutkintotyössä selvitetään sitä, miten ohjelmistopakettien kokoajan työtoimenkuvaa voidaan helpottaa automatisoinnin avulla ja sitä, mistä konfiguraationhallinta koostuu.

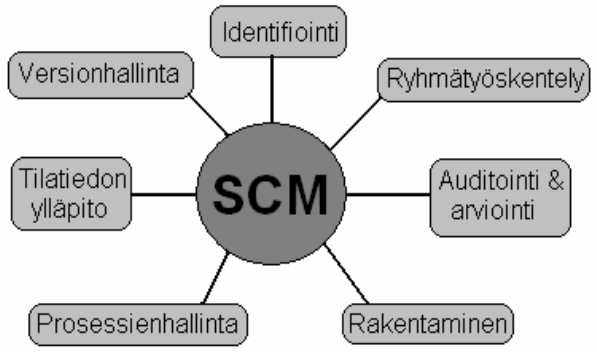
Tästä tutkintotyöstä on hyötyä varsinkin juuri koulusta valmistuneelle ohjelmistoalan henkilölle, joka suunnittelee työskentelevänsä ohjelmistopakettien kokoajana tai työskentelee ohjelmistojen konfiguraationhallinnan osa-alueella.

Tämän tutkintotyön runko koostuu konfiguraationhallinnan toiminnasta, jatkuu ohjelmistopakettien kokoajan työtoimenkuvasta ja työkaluista ja päättyy ohjelmistopakettien kokoamisen automatisointiin.

2 OHJELMISTOTUOTTEEN KONFIGURAATIONHALLINTA

Ohjelmistojen kehityksessä käytetään monia erilaisia sääntöjä ja määritelmiä, joita noudattamalla saadaan työpanokseen nähden paras mahdollinen työtulos. Tällaista menetelmää kutsutaan ohjelmistotuotteen konfiguraationhallinnaksi (engl. software configuration management (SCM)). Sen tarkoitus on estää virheiden syntymistä erilaisin keinoin ja vahtia virheiden välttämistä koko ohjelmiston elinkaaren ajan. Konfiguraationhallintaa käytetään yleisesti ohjelmistofirmoissa, jotka ovat erikoistuneet ohjelmistojen tekemiseen. Erilaisia konfiguraationhallinnan menetelmiä on monenlaisia ja ne on yleensä rakennettu juuri yhtiön omien tarpeiden mukaisesti.

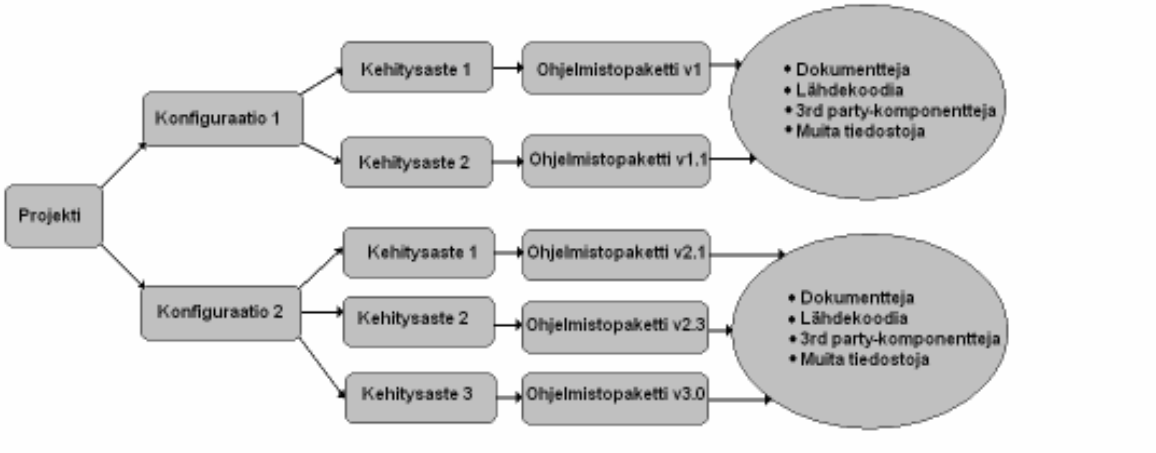
2.1 Konfiguraationhallinnan osa-alueet



Kuva 1 Konfiguraationhallinan tärkeimmät osa-alueet

IEEE standardissa (ANSI/IEEE 1042 - 1987) kuvan 1 mukaisesti konfiguraationhallinta sisältää identifioinnin, versionhallinnan, tilatiedon ylläpidon sekä auditoinnin ja arvioinnin. Näiden osa-alueiden lisäksi on joihinkin

konfiguraatiojärjestelmiin lisätty myös rakentamisen, prosessienhallinnan ja ryhmätyöskentelyn osa-alueet. /1, s. 9; 2/



Kuva 2 Projektihierarkia

Kuvan 2 mukaisesti konfiguraationhallinta koostuu projektista, jolle luodaan konfiguraatio. Jokaiselle konfiguraatiolle on oma kehitysasteensa, jonka mukaan projekti etenee. Kun kehitysaste on saatu päätökseen, tehdään siitä ohjelmistopaketti. Ohjelmistopaketti sisältää kaikki projektin käännetyt materiaalit kyseisestä kehitysasteesta.

2.1.1 Identifiointi

Identifioinnissa kuvataan ohjelmistotuotteen rakenne ja nimetään siihen kuuluvat komponentit ja niiden tyypit. Tästä saadaan yksilöity sovellus ja sovelluskomponentit, joihin voidaan viitata muista järjestelmän osista ja kuvauksista. /1, s. 1/

Identifioinnin osa-alueen tarkoitus on todentaa se, milloin muutokseen on tarvetta. Jos esimerkiksi jokin ulkopuolinen komponentti muuttuu olennaisesti, täytyy

identifioida muutoksen tarvetta myös itse ohjelmaan. Mikäli muutos tehdään, syntyy tästä oma uusi versionsa.

Projekteihin voi syntyä muutoksia valmiiden spesifikaatioiden jälkeen mm. uusista muutoksista spesifikaatioihin tai erilaisista virheistä eli bugeista.

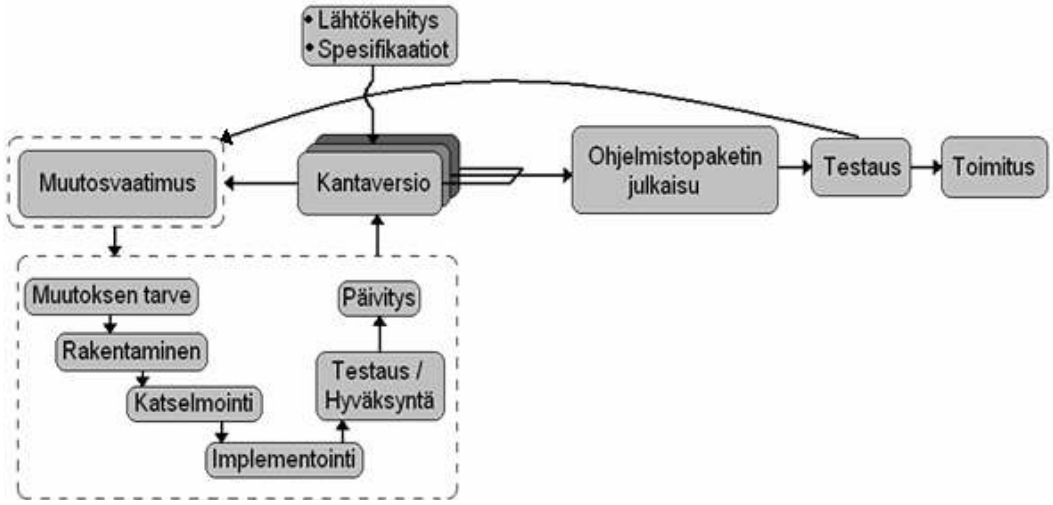
2.1.2 Versionhallinta

Ohjelmistojen muutosten hallinnan tuloksena syntyy ohjelmistoista monia erilaisia julkaisuversioita. Versionhallinnassa varmistetaan se, ettei päällekkäisessä päivityksessä synny virhetilanteita ja se, että ohjelmisto on yhtenäinen luomalla ns. perustuote (engl. baseline product).

Kantaversio (engl. baseline) on konfiguraationhallinnassa perusta, jonka mukaan työtä kehitetään eteenpäin. Kantaversio luodaan ohjelmistoprojektin alussa ja jäädytetään, kun toiminnallinen runko on todettu toimivaksi. Kantaversiota kehitetään asettamalla tavoitteet tietyn ajan sisälle ja lisäämällä tänä aikana kantaversioon uusia toiminnallisuuksia eli muutoksia. Uusista lisätyistä muutoksista muodostuu vanhan kantaversion rinnalle uusi kantaversio, joka sisältää vain sillä hetkellä lisätyt muutokset.

Kun kantaversio sisältää projektin tuotteelle asetetut vaatimukset, luodaan siitä julkaisuversio. Ennen virallista julkaisua tuote testataan tärkeimmiltä toiminnallisuuksiltaan, ja mikäli kaikki toiminnallisuudet todetaan toimiviksi, tehdään tuotteelle virallinen julkaisu ja toimitus (engl. release and delivery).

Julkaistun tuotteen versio jäädytetään ja siitä muodostuu uusi kantaversio. Täten uusi versio voidaan luoda käyttämällä vanhaa julkaisua pohjana.



Kuva 3 Kantaversion kehityskaari

Kuvan 3 mukaisesti, kun kantaversioon lisätään uutta toiminnallisuutta, luodaan uusi muutosvaatimus. Ensimmäisenä varmistetaan, että muutos on tarpeellinen ja spesifikaation mukainen. Kun muutoksen tarve on todettu, kehittäjät tekevät muutoksen lähdekoodiin. Valmis koodi katselmoidaan ja testataan, minkä jälkeen uusi toiminnallisuus lisätään versionhallintatyökalun avulla tietokantaan.

2.1.3 Tilatiedon ylläpito

Ohjelmistokehityksen aikana erilaisilla komponenteilla voi olla monenlaisia työtiloja, esimerkiksi työn alaisena, testattava, SQA, integroitava, jäädytetty ja valmis. Tilatieto helpottaa työn valmistumista ja selventää, mitä työvaiheita on tehty ja mitä on tekemättä. Tämä osa-alue siis kontrolloi muutoksia.

Työn alaisena oleva tilatieto kertoo muille kehittäjille, että tiettyä osaa ohjelmakoodista kehitetään, muokataan tai korjataan parhaillaan. Tällöin voidaan

kehittää ohjelmakoodia ainoastaan samanaikaisena kehityksenä, joka täytyy lopuksi yhdistää konfliktien varalta. Yhdistämisessä jälkimmäisenä oleva päivittäjä joutuu tekemään yhdistämisen oman ja aikaisemman päivittäjän lähdekoodien välillä. Ohjelmistopakettien kokoajan tehtäviin kuuluu myös seurata yhdistämisiä, jotta saadaan varmuus sulautumisen onnistumisesta.

Testattavana oleva ohjelma sisältää usein paljon uutta tai integroitua ohjelmakoodia. Toimivaksi todetun ja testatun ohjelmakoodin jälkeen siirrytään SQA tilaan (engl. software quality assured). Jotta ohjelma voi siirtyä testattavaksi, joudutaan käymään läpi integraatiotila, jossa ohjelmaa kootaan pienistä osasista toimivaksi kokonaisuudeksi. Tällöin uusia muutoksia ei hyväksytä mukaan, ellei kyseessä ole kriittinen muutos tai ohjelman toiminnan korjaus.

Kun ohjelman kehitys on saatu tarpeeksi pitkälle ja testaukset todettu toimiviksi, siirrytään ohjelmassa jäädytystilaan. Tässä tilassa estetään kaikki uudet muutokset jo tehtyyn lähdekoodiin. Jäädytystilaan voidaan siirtyä vain, jos ohjelma on aikaisemmin ollut SQA tilassa. Kun kehitys tietyn version osalta on saatu loppuun, siirrytään valmistilaan. Valmistilassa oleva ohjelma on julkaisukelpoinen.

2.1.4 Auditointi ja arviointi

Ohjelmistotuote on validoitava, jotta saadaan varmuus sen toimivuudesta. Tällöin tarkastetaan, että tehdään oikeaa tuotetta ja se, että tuotetta tehdään oikein (validointi & verifiointi -tekniikka).

Jotta saataisiin täysi varmuus siitä, että muutokset toteutetaan oikein ja asianmukaisesti, täytyy varmistaa ohjelmistotuotteeseen liittyvien kuvausten oikeellisuus ja riittävyys. Auditoinnissa kolmas osapuoli suorittaa

laaduntarkistuksen ohjelman rungolle ja etsii lähdekoodista mahdollisia virheitä. Tämän tarkoituksena on pyrkiä säilyttämään yhtenäinen komponenttikokoelma.

Julkaisuversioista pidetään yleensä aina oma katselmus, jossa katsotaan läpi ohjelman toiminnallisuudet ja mahdolliset virheet. Tämän vaiheen voi liittää myös ohjelman testausvaiheeseen. Mikäli katselmuksessa todetaan virheitä tai toiminnallisia puutteita, korjataan ne ensisijaisesti, jotta julkaisuversion voi vapauttaa levitykseen.

2.1.5 Rakentaminen

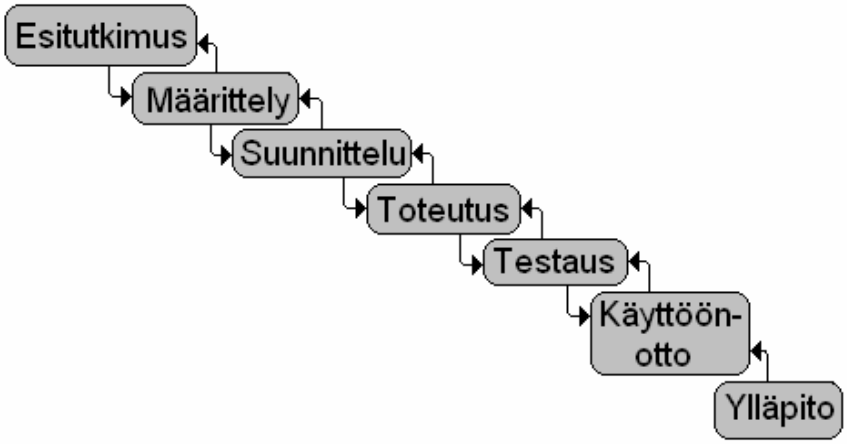
Rakentamisen osa-alueessa pyritään ohjelmistotuote toteuttamaan parhaalla mahdollisella tavalla ja varmistamaan tuotteen rakenteen hallinta. Jotta rakentaminen olisi mahdollisimman yksinkertaista ja pohjustettua, dokumentaation on oltava mahdollisimman tarkkaa ja täydellistä.

Ohjelmiston rakenne sisältää usein käyttöliittymän, tietokantoja, graafisuutta ja tukea ulkopuolisille komponenteille. Rakentamisessa keskitytään näiden yhteiseen sulauttamiseen eli integrointiin. On myös tärkeätä priorisoida ja jakaa rakentaminen mahdollisimman loogisesti ja selkeästi. Käyttöliittymän kehittäminen turvalliseksi ja aukottomaksi luo ohjelmistolle laadukkuutta.

Rakentamisessa pyritään pitämään ohjelmiston rakenne selkeänä, virheettömänä ja monikäyttöisenä. Tämä mahdollistaa uusiokäytön, joka säästää ohjelmistoprojektien aikaa vastaavanlaisissa töissä. Uusiokäytössä on myös omat riskinsä, kuten epävarmuus toiminnasta erilaisissa ympäristöissä tai versioissa. Pitämällä dokumentaatio tarpeeksi ymmärrettävällä tasolla, uusiokäytön riskiä pystytään pienentämään.

2.1.6 Prosessienhallinta

Prosessienhallinnassa pyritään noudattamaan organisaation omia työ- ja toimintatapoja ja prosessille määrättyä elinkaarta. Prosessin elinkaaren voi kuvata vesiputousmallina, joka sisältää kaikki prosessin työvaiheet. Vesiputousmalli on kuitenkin vain ideaalinen tapa, jota harvoin pystyy noudattamaan siirtymävaiheiltaan. Kuvan 4 mukaisesti vesiputousmallissa työvaihe siirtymän rajapinta on tiukasti sidottu edellisen ja seuraavan kanssa. Jos esimerkiksi projektissa joudutaan keskeyttämään kehitys, on taloudellisempaa alkaa kokonaan alusta kuin tehdä monia korjauksia hylättyyn ratkaisuun.



Kuva 4 Vesiputousmalli

2.1.7 Ryhmätyöskentely

Ryhmätyöskentelyn osa-alueessa varmistetaan se, että tuotteen yhtäaikaisten kehittäjien tekemä työ ja heidän välisensä vuorovaikutus on hallittua. Tämä hoidetaan yleensä antamalla jokaiselle työntekijälle oma tehtävänsä. Kun tehtävää aletaan tehdä, siirtyy muokattavan komponentin tila työskentelytilaan, jolloin muut työntekijät eivät pysty tekemään päällekkäisiä muutoksia työn alla oleviin

tiedostoihin. Tilatiedon ylläpito on suunniteltu juuri ryhmätyöskentelyn osa-alueen avuksi.

Vaikka päällekkäistä työn tekemistä pyritään välttämään, joissain projekteissa saatetaan käyttää yhtäaikaista ohjelmakehitystä (engl. parallel development). Tällaisia tapauksia syntyy yleensä silloin, kun samanaikaisesti aloitetaan kehittämään uutta versiota vanhan rinnalla. Myöhemmin nämä kaksi versiota voidaan yhdistää toisiinsa (engl. merge).

Ryhmätyöskentely sisältää myös käytännön toimintatavat neuvottelutilaisuuksissa. Projektien eteneminen edellyttää sitä, että työntekijät tietyin väliajoin kokoontuvat neuvottelemaan projektin etenemisestä. Työntekijän on varauduttava suurissa yrityksissä siihen, että kokoukset saattavat olla videoneuvotteluja ja monesti englanniksi, sillä harvoin kaikki työntekijät puhuvat äidinkielenään samaa kieltä tai asuvat samassa maassa. Hyvä kommunikointi on ryhmätyöskentelyssä ensisijaisen tärkeätä.

3 OHJELMISTOPAKETIN RAKENNUKSEN HALLINTA

Ohjelmistopaketin valmistaminen valmiista lähdekoodeista ja komponenteista on monivaiheinen prosessi. Tämän prosessin keskeisimmät työntekijät ovat ohjelmistopakettien kokoaja ja ohjelmistopakettien julkaisija. Heidän vastuullaan on ohjelmistopaketin toimittaminen (engl. delivery) eli valmiin paketin lähetys julkaistavaksi.

Ennen kuin ohjelmistopakettia voidaan lähteä kokoamaan, tarvitaan tähän oikeanlainen ympäristö. Ohjelmistopaketin kokoajan tehtävä onkin luoda tämä

ympäristö ja oikeilla työkaluilla hakea valmiit komponentit ja lähdekoodit ympäristöön. Ohjelmistopaketin kokoaja kokoaa paketin hallitusti ja varmistaa tuotteen toimivuuden ennen kuin antaa paketin ohjelmistopaketin julkaisijalle julkaistavaksi.

3.1 Ohjelmistopakettien kokoajan työtoimenkuva

Päätehtävänään ohjelmistopaketin kokoaja kokoaa ohjelmistopaketit ja julkaisee ne ohjelmistopaketin julkaisijan kanssa. Ennen ohjelmistopaketin kokoamista on ohjelmistopaketin kokoajan käytävä läpi monia eri työvaiheita. Työnkuvaan kuuluu mm. versionhallintaa, ohjelmistopakettien kääntäminen toimituspaketiksi, käännösvirheiden seuraaminen, virheiden ilmoittaminen eteenpäin, ohjelmiston rakenteen organisointi, kantaversioiden luominen, automatisoinnin kehittäminen, raporttien sekä julkaisutiedotusten luominen ja erilaisten konfiguraatioiden asettaminen.

Kantaversioita luodessaan ohjelmistopakettien kokoaja jäädyttää ohjelmiston julkaisuja tietyin väliajoin. Tällöin ohjelma joko testataan tai julkaistaan. Kun kantaversio luodaan, projektin hierarkia jäädytetään senhetkiseen tilaansa ja yleensä tämän jälkeen vähintään testataan yrityksen sisäisesti. Tämä mahdollistaa sen, että ohjelmistojen kehittäjien työn tulokset voidaan katselmoida, ja kehittäjät saavat käyttöönsä aina viimeisimmät muutokset.

Ohjelmistojen rakenteen ja elinkaaren organisointi kuuluu osittain myös ohjelmistopakettien kokoajan työnkuvaan. Uusien projektien syntyessä ohjelmistopakettien kokoaja integroi muutokset projekteihin, sekä kääntää ja testaa lähdekoodin kehittäjien tekemät muutokset. Työnkuvaan kuuluu myös seurata lähdekoodin tekijöiden rinnakkaista kehittämistä ja potentiaalisia konflikteja.

Ohjelmistopakettien kokoaja varmistaa myös sen, että lähdekoodin kehittäjät yhdistävät lähdekoodinsa oikein konfliktitilanteissa.

Ohjelmistopaketin kokoajan on pystyttävä kommunikoimaan työssään hyvin, sillä käännösvaiheessa syntyy usein virheitä ja niiden raportointi on tehtävä mahdollisimman tarkasti, jotta virhe tulee varmasti korjatuksi. Jotta toimituspakettien kokoaminen onnistuisi mahdollisimman hyvin, on ohjelmistopaketin kokoajan ymmärrettävä käännettävän ohjelmiston rakenne täysin. Monesti lähdekoodin kehittäjät ja testaajat tarvitsevat ohjelmistopakettien kokoajan asiantuntemusta konfiguraation- ja versionhallinnassa. Koska ohjelmistopaketin kokoaja työskentelee ohjelmistoprojektin loppupäässä, julkaisupaineet kohdistuvat täten suoraan häneen. Tästä syystä hänen on ohjelmistoprojekteissa pystyttävä kestämään ympärillä syntyvää painetta ja tekemään oikeita päätöksiä nopeasti.

Työkalut vaihtuvat usein uuden teknologian käyttöönotossa. Tämän ansiosta ohjelmistopaketin kokoajat ovat yleensä ensimmäisinä käyttämässä uusia käännöstyökaluja. Tästä syystä on tärkeätä opetella jatkuvasti uusien toimintojen käyttöä. Oikein käytetyt työkalut antavat usein entistä paremman työtuloksen laadukkuutena.

3.2 Menettely virhetilanteissa

Mikäli ohjelmiston käännöksessä syntyy virheitä tai varoituksia, täytyy näiden korjaamiseksi toimia virheenhallinnan (engl. error management) mukaisesti. Ensisijaisesti on tarkasteltava, vaikuttaako virhe olennaisesti ohjelman kääntymiseen tai toiminnallisuuteen. Mikäli niin on, pitää virheen syntykohta pystyä jäljittämään.

Käännöstyökalut kirjoittavat usein lokitiedostoja, jotka sisältävät yksityiskohtaisesti kaikki käännöksen aikana tapahtuvat toimenpiteet. Usein ensimmäinen virhe synnyttää käännösvaiheessa joukon muitakin virheitä. Jotta oikea virhe saadaan löydettyä, täytyy lokitiedostoja seurata loogisessa järjestyksessä, joka yleisesti on toimenpiteen tai tiedoston aikaleimauksen mukainen.

Virheen löytyessä raportoidaan sen esiintymiskohta, minkä jälkeen kehittäjä tekee lähdekoodiin päivityksen. Päivityksen jälkeen käännetään uudestaan kaikki komponentit, joihin virhe on vaikuttanut. Monimutkaisimmissa projekteissa kannattaa kuitenkin kääntää kaikki mahdolliset komponentit uudestaan siltä varalta, että virhe on vaikuttanut myös muiden komponenttien toiminnallisuuteen. Tällaisissa tapauksissa on huomioitava se, että kääntäminen ylikirjoittaa vanhojen tiedostojen päälle. Näin ollen esimerkiksi lokitiedostot on hyvä ottaa talteen ennen uuden käännöksen aloittamista.

Virhetilanteiden syntyessä suurissa projekteissa pyritään myös priorisoimaan virheet. Virhetilanteet on usein luokiteltu A – E -luokkiin. A- ja B-luokan virheet ovat ns. ”show stopper” -virheitä, jotka ovat katastrofaalisia ja omaavat korkeimman prioriteetin korjaukseen. Yleensä A-luokan virheiden korjaus kestää usean päivän. Vähemmän haitallisia näistä kahdesta ovat B-luokan virheet. Nämä ovat kriittisiä virheitä, jotka yleensä myös pysäyttävät ohjelman toimimisen jo alkuvaiheessa. C-luokan virheet ovat vakavia, mutta yleensä ne liittyvät vain tiettyyn toiminnallisuuteen. Pienimpiä prioriteetteja omaavat D- ja E-luokat. D-luokan virhe on vähäinen ja aiheuttaa satunnaisia virheitä. E-luokan virhe ei sisällä ohjelman toimintaan vaikuttavia virheitä. Tällaisia ovat esimerkiksi kuvien puuttumiset, toiminnallisuuden kuten painonappien väärin sijoittelu ja kirjoitusvirheet.

3.3 Ohjelmistopaketin kokoajan työkalut

Ohjelmistopaketin kokoajan työssä on käytössä suuri määrä erilaisia työkaluja, joita tarvitaan käännösten tekemisessä ja testauksessa. Keskeisimpiä työkaluja ovat versionhallintatyökalut, vertailutyökalut, käännösohjelmistot, testaustyökalut ja erilaiset tekstieditorit.

Työkalujen oikeanlainen käyttö lisättynä oikeanlaiseen projektin toteuttamiseen takaa ohjelmistopaketin laadukkuuden. Ohjelmistopakettia koottaessa työkaluilla voidaan varmistaa paketin toiminta monissa erilaisissa järjestelmissä. Ohjelmistopakettien kokoajan suurin työ onkin pysyä ajan tasalla uusien työkaluversioiden osalta, jotta hän osaa käyttää niitä juuri niin kuin niitä on tarkoitettu käytettävän. Uuden teknologian tullessa usein myös käännöstyökalut vaihtuvat.

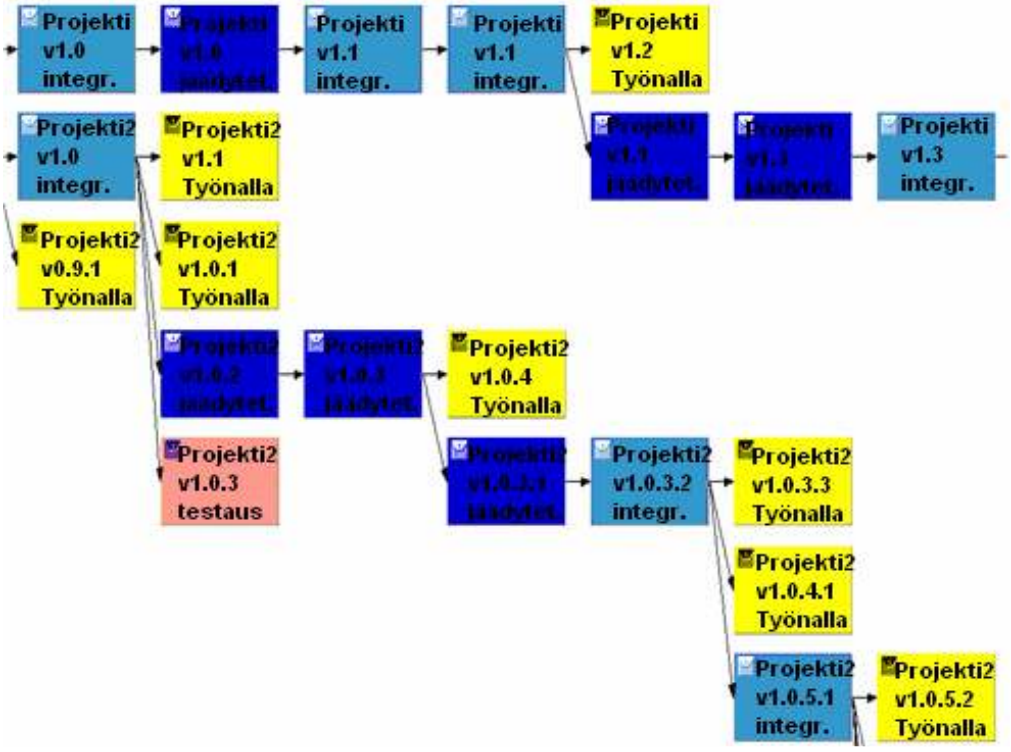
3.3.1 Versionhallintatyökalu

Versionhallintatyökalu on konfiguraationhallinnassa tärkein projektin muutostenhallinnan ylläpitotyökalu. Projektin edetessä syntyy dokumentteja, lähdekoodia ja muuta materiaalia. Ne pyritään säilyttämään yleensä yhdessä paikassa, jota ylläpidetään erilaisilla versionhallintatyökaluilla.

Näillä työkaluilla hallinnoidaan ohjelman perusrunkoa eli kantaversiota. Kantaversio sisältää esimerkiksi ohjelmistotuotteen lähdekoodit. Kun ohjelmistotuotteen kantaversio on luotu ja jäädytetty, aletaan tällä työkalulla kehittää edelleen tuotetta. Versionhallintatyökalulla jokaisesta muutoksesta, jonka kantaversio saa, luodaan oma delta-versio. Kun kehitys on tarpeeksi pitkällä ja delta-versio on katselmoitu, tehdään viimeisimmästä delta-versiosta uusi

kantaversio. Näin kehittäjät saavat käyttöönsä aina uusimman version työalustakseen. /3, s. 4/

Versiotyökaluilla pidetään yllä tietoa ohjelmistoprojektien yksityiskohtaisista tiedoista aina siitä lähtien, mitä eri versiot sisältävät ja ketkä ovat muunnelleet lähdekoodia projektin missäkin vaiheessa ja millaisessa tilassa mikäkin versio on. Kuvan 5 mukaisesti versiohistoriasta näkee yleensä nopealla silmäyksellä yleiskuvan kaikista versioista ja haarautumista. Haarautumat voivat syntyä esimerkiksi eri alustoille tehtävissä käännöksissä tai erilaisten komponenttien käyttämisestä. Monet versionhallintatyökalut toimivat pohjana ohjelmistokehitykselle ja tämän takia niiden käyttämisessä ja päivittämisessä täytyy olla varovainen. /4/



Kuva 5 Esimerkkikuva erään versionhallintaohjelman versiohistoriasta

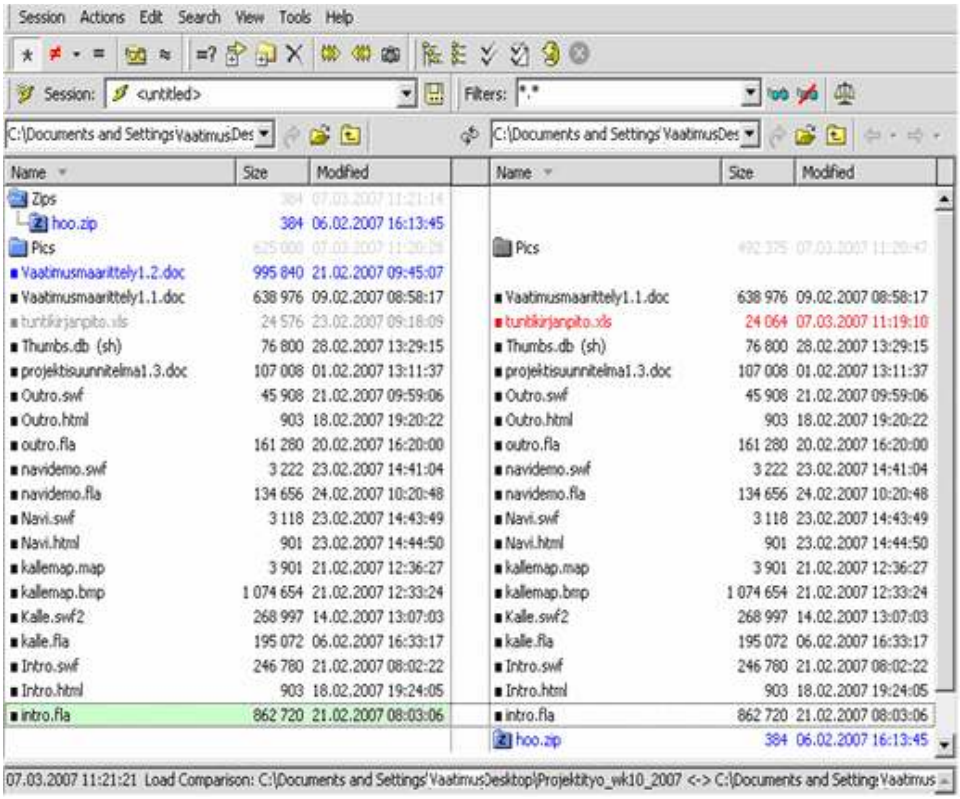
Versionhallintatyökaluja on monenlaisia eri tarkoituksiin ja niistä löytää lisätietoja viitteistä /5/ ja /6/.

3.3.2 Vertailutyökalu

Vertailutyökalulla (engl. compare-tool) voidaan nopeasti tarkastella versioiden välisiä muutoksia. Versioiden vaihtuessa hyvä työkalu pystyy osoittamaan sen, mikä tiedosto on muuttunut ja miten se eroaa aikaisemmasta versiosta. Vertailutyökalu tarvitsee siis vähintään kaksi tiedostoa tai hakemistoa, jotta vertailu onnistuu. Muutokset tiedostojen välillä ovat yleensä värikoodattuja, joten käyttäjän on helppo seurata, mitä uuden ja vanhan version välillä on tapahtunut.

Tavanomaisimpia muutoksia ovat lisätyt ja korjatut rivit sekä tiedostot. Mikäli jokin tiedosto tai hakemisto puuttuu, näytetään se omana värinään. Hyvissä vertailutyökaluissa on myös muunlaisia ominaisuuksia, kuten muutoksien kopiointi, tiedostojen ja hakemistojen muokkaus, automaattinen muutosten seuraus ja tiettyjen tiedostojen filtteröinti, joiden muutoksia ei haluta seurata.

Vertailutyökalua käytetään usein virhetilanteissa, joissa aikaisempi versio on toiminut, mutta päivitetty uudempi versio ei jostain syystä toimi. Tällöin vertaillaan näiden kahden versioiden välisiä eroja ja etsitään kohta, joka estää kääntämisen. On kuitenkin hyvä muistaa, että jotkut ohjelmistot voivat sisältää suuren määrän tiedostoja ja hakemistoja, joten tällaisissa tapauksissa vertailuoperaatio on raskas prosessi tietokoneen suorituskyvylle ja erityisesti massamuistin käytölle. Tämän johdosta onkin hyvä filtteröidä kaikki binäärikooditiedostot vertailusta ja jättää vain selväkieliset tiedostot vertailtaviksi. Kuvassa 6 on esimerkki siitä, millainen yleisesti on vertailutyökalun hakemistopuu.



Kuva 6 Esimerkkikuva eräästä vertailuohjelmasta

3.3.3 Käännösohjelmisto

Ohjelmistopakettien tekeminen binääritiedostoksi eli tietokoneen ymmärrettäväksi käännökseksi onnistuu monenlaisella kääntäjällä. Käännösohjelmisto valitaan sen mukaan, millä kielellä lähdekoodi on tehty.

Käännösohjelmisto tarjoaa yleensä tekstieditorin lähdekoodin kehittämiseen ja debuggerin käännettävän koodin virheen jäljitykseen. Suuret ohjelmistoyritykset panostavat yleensä yhteen kääntäjään, joka valitaan useimmiten sen helppokäyttöisyyden ja tehokkuuden perusteella.

Käännösohjelmistoilla luodaan usein ohjelman lähdekoodin kehitys, mutta itse valmiin ohjelmistopaketin luominen tehdään yleensä komentorivipohjaisesti. Tällöin voidaan käyttää apuna erilaisia käännös-skriptejä, jotka automatisoivat ja helpottavat käännöstä.

Suosituimmat käytössä olevat kaupalliset C++ -käännösohjelmistot ovat Microsoftin Visual Studio ja Borlandin C++. Ilmaisista kääntäjistä suosituin on Bloodshed Softwaren Dev-C++.

3.3.4 Lähdekoodin toiminnallisuuden testaustyökalu

Erilaisia lähdekoodin toiminnallisuutta testaavia työkaluja on erittäin paljon ja tietyille koodikielille ja osa-alueille, joten on tärkeätä tietää mitä ohjelmakoodin osa-alueita on testattava. Tavanomaiset testausalueet ovat turvallisuus ja suojaus, oikeanlainen toiminnallisuus ja tehokkuus. Näistä syntyy ohjelman laadukkuus, jota ohjelmistoyritykset ja ohjelmien käyttäjät arvostavat.

Ohjelmakoodia testattaessa täytyy priorisoida testattavat osa-alueet. Turvallisuuteen on hyvä panostaa, jotta ohjelmaa ei pysty käyttämään siten, miten sitä ei ole tarkoitettu käytettäväksi. Myös ohjelman toiminnallisuuden selkeys on tärkeää, jotta ohjelma tekee juuri sitä, mitä käyttäjä olettaa ja haluaa sen tekevän.

3.3.5 Tekstieditori

Ohjelmiston rakentamisvaiheessa syntyy monesti erilaisia virhetilanteita. Nämä saattavat johtua esimerkiksi skripteihin väärin asetetuista lipuista, joiden mukaan

kääntäjä tekee tietynlaista käännöstä. Jotta liput voidaan asettaa oikeiksi, tarvitaan siihen jonkinlaista editoria, jolla skripti-tiedostoa pystyy muokkaamaan.

Tekstieditoreja on monenlaisia ja parhaimpiin on lisätty monenlaisia apukeinoja nopeuttamaan tiedostojen käsittelyä ja muokkausta. Näitä on mm. editorien omat skriptit, joilla voidaan nopeuttaa ja automatisoida eri toimintoja. Esimerkiksi tiettyjen sanojen värikoodaus lukemisen selkeyttämiseksi, rivitys ja sisennys ja pikavalinnat tietyille usein käytetyille sanoille helpottavat ohjelmoijan työtä seurata lähdekoodia.

Tunnetuin tekstieditori Windows- käyttöjärjestelmälle on Notepad, joka tulee käyttöjärjestelmän mukana, mutta tämä editori ei sisällä paljoa erityisominaisuuksia. Notepad++, Scite, UltraEdit ja Editpad ovat myös Windows-käyttöjärjestelmässä toimivia tekstieditoreita, mutta nämä sisältävät jo paljon toiminnallisuutta tavallisen tekstieditoinnin lisäksi. Unix ja Linux käyttöjärjestelmissä suosituin editori on Emacs ja Vim, jotka molemmat sisältävät erittäin paljon toiminnallisuutta. Kevyempiä editoreita tälle käyttöjärjestelmälle ovat esimerkiksi nano ja pico, joissa on perustoiminnallisuudet tiedoston sisällön muokkaukselle.

4 OHJELMISTOPAKETTIEN KOKOAMINEN

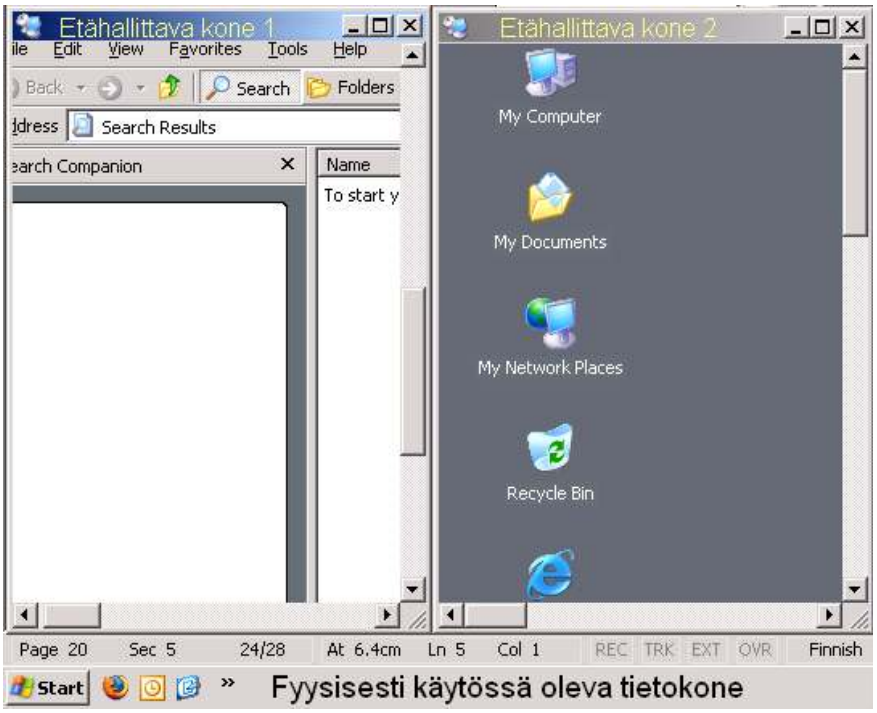
Ohjelmistopakettien kokoaminen tapahtuu ohjelmistoprojektien viimeisenä työvaiheena, kun kehittäjät (engl. developer) ovat saaneet lähdekoodin valmiiksi. Ohjelmistopaketin kokoamisen suorittaa ohjelmistopakettien kokoaja, jonka ensimmäinen työtehtävä on koota käännösympäristö kaikelle tarvittavalle käännettävälle materiaalille. Käännösympäristön kokoamisen jälkeen suoritetaan usein erilaisia tarkastuksia, jotta saadaan varmuus ympäristön oikeanlaisuudesta.

Projektin koosta riippuen, pelkkä materiaalin kokoaminen voi viedä tunteja ja kuluttaa suuresti käytettävissä olevaa verkkokaistaa. Kun ympäristö on varmennettu, voidaan käännös aloittaa. Jos käännös on saatu virheettömästi luotua, ohjelmistopakettien kokoaja tai oma testiryhmä testaa käännöksen läpi. Mikäli testivaihe menee virheettömästi läpi, ohjelmistopakettien kokoaja ilmoittaa toimivuuden ohjelmistopakettien julkaisijalle, joka lähettää valmiin paketin eteenpäin julkaistavaksi ja toimitettavaksi.

4.1 Käännösympäristön kokoaminen

Kun lähdekoodin valmistumisesta on annettu ilmoitus ohjelmistopakettien kokoajalle, täytyy seuraavaksi hakea käännökseen tarvittava sisältö. Kokoonpanon hallinnassa materiaali haetaan replikoimalla tietokantojen välillä. Replikointi ottaa palvelimelta kopion käännettävästä versiosta omaan työhakemistoon ja yleensä haetaan vain edellisen replikoinnin jälkeen muuttuneet tiedostot. On myös hyvä tietää millaisella kokoonpanolla työtä käännetään ja se, että koneessa on tarpeeksi resursseja käännöksen suorittamiseksi. Suuret ohjelmistoprojektit voivat pitää sisällään monia gigatavuja käännettävää materiaalia.

Käännöskoneet ovat yleensä hyvinkin tehokkaita ja kalliita, joten jokainen ohjelmistopakettien kokoaja ei välttämättä pysty saamaan omaa konetta fyysisesti käyttöönsä. Kuvan 7 esimerkin mukaisesti käännösympäristöt luodaan suurissa ohjelmistoprojekteissa käännöskoneilla etähallinnan (engl. remote desktop) avulla, jossa kokoonpanija kokoaa käännösympäristönsä eri koneeseen, kuin on itse fyysisesti käyttämässä. Tässä tilanteessa on huomioitava se, että etähallitulla koneella voi olla muitakin käännöksiä ajossa samanaikaisesti.



Kuva 7 Kuvakaappaus kahdesta etähallittavasta käännöskoneesta

Etähallitulla koneella on hyvä pitää pelisäännöt siitä, millä asemilla ja millaisissa hakemistopoluissa materiaalia säilytetään. Tämä sen vuoksi, etteivät eri käännösten materiaalit mene sekaisin keskenään. Kansiot onkin hyvä nimetä vähintään päivämäärän ja tekijän nimen mukaan, jolloin sekaannuksia ei pääse helposti syntymään.

Kun materiaali on saatu koottua, täytyy varmistua siitä, että käännöstyökalut ovat päivitetty viimeisimpään versioonsa. Etähallituissa koneissa ei yleensä tarvitse itse päivittää työkaluja, sillä tämä tehdään automaattisesti. Itse päivittämiseen liittyy muutenkin riski siitä, että toisella ohjelmistopakettien kokoajalla jo käännöksessä oleva työ saattaisi mennä rikki, kun työkalut muuttuvat kesken käännöksen. Kun työkalut ja käännösalue on kunnossa, on työ valmis käännettäväksi.

4.2 Käännöksen toteuttaminen

Käännöksen luominen toteutetaan monesti komentoriviltä. Yleensä suurien projektien käytössä on oma työkaluryhmä, joka kustomoi ja optimoi työkalut, sekä tekevät automatisointi-skriptit helpottamaan käännöstyötä mahdollisimman paljon. Skriptien on yleensä hankala tutkia missä hakemistopoluissa materiaali sijaitsee, joten yleensä käytetään subst-komentoa Windows-ympäristöissä muuttamaan hakemistopolku tietyksi virtuaaliseksi työasemaksi. Skriptien on helpompaa toimia silloin, kun hakemistojuuri toimii aina samoilla kansiolla.

Manuaalista käännöstä käytetään erittäin harvoin, sillä skriptit on todettu erittäin hyödyllisiksi käännöksissä. Skripti voi olla tavallinen tiedosto, joka sisältää käännöskomentoja järjestyksessä riveittäin. Yleensä skriptit pitävät vielä sisällään virheentarkistusta, mikäli käännöksen aikana tapahtuva virhe tai varoitus on kriittinen. Näin ei tarvitse odottaa käännöksen loppuun asti ja todeta käännöksen olleen rikki, vaan aikaa säästäen voi lähettää tiedon rikkinäisestä osasta eteenpäin ja koota päivitetty käännösympäristö korjatulla materiaalilla.

Ohjelmisto saattaa usein rakentua monivaiheisesti. Tällaisia vaiheita on esimerkiksi esikäännös (engl. prebuild), pääkäännös (engl. mainbuild) ja jälkikäännös (engl. postbuild). Esikäännösvaiheessa tehdään ennen pääkäännöstä tehtävä asioita, kuten työkalujen ja konfiguraatioiden siirtoa. Myös asetukset tarkastetaan tässä vaiheessa kuntoon. Pääkäännöksessä käännetään kaikki mahdolliset komponentit, joita ohjelmistopakettiin halutaan mukaan. Jälkikäännöksessä käännetään kaikki, mitä normaalin käännöksen aikana ei vielä pysty kääntämään, kuten valmiit binääripaketit. Jälkikäännöstä käytetään usein myös siksi, että pääkäännöksen ollessa virheellinen, ei tarvitse aikaa säästääkseen yrittää tehdä jälkikäännöstä.

4.3 Valmiin ohjelmistopaketin testaaminen

Kun käännös on valmis, tarkistetaan lokitiedot käännöksestä. Mikäli kaikki näyttää olevan kunnossa, testataan paketti erilaisilla testeillä. Näitä voivat olla mm. sauhu-testi, käynnistys-testi ja läpiajo-testi.

Sauhu-testissä pyritään testaamaan paketin ja laitteiston yhteensopivuutta. Tällöin käytettävää laitteistoa kuormitetaan ja varmistetaan se, ettei ohjelma kaadu kovassakaan käytössä. Sauhu-testin ensisijainen tarkoitus on todentaa, ettei ohjelma saa sellaisia virheitä perustoiminnassa, joita julkaisuversiossa ei saa olla. Käynnistys-testissä testataan vain, että ohjelmisto käynnistyy (engl. boot). Läpiajotestissä ajetaan ohjelmaa läpi ja pyritään saamaan mahdolliset virheet näkyviin. Samalla katsotaan ohjelman looginen toimintajärjestys läpi. Läpiajo-testiä kutsutaan toisinaan myös stressi-testiksi. Nämä yllämainitut testit tehdään yleensä testi- ja verifiointivaiheessa.

4.4 Ohjelmistopaketin toimittaminen eteenpäin

Kun ohjelmistopaketti läpäisee vielä kaikki sille asetetut testit, on se valmis toimitettavaksi eteenpäin. Tämän osuuden hoitaa ohjelmistopakettien julkaisija, joka on yleensä vastuussa siitä, mitä ohjelmistopakettien kokoaja työssään tekee.

Valmiista ohjelmistopaketin käännöksestä on tehtävä vielä yhteenveto, joka sisältää tiedot käännetyistä osista ja komponenteista, jotka paketti pitää sisällään. Tähän on vielä hyvä lisätä mille laitteelle paketti on todettu toimivaksi ja mikä versio paketista ja laitteesta on kyseessä. Helpoin tapa toimittaa käännetty ohjelmistopaketti eteenpäin on luoda käännöksestä pakattu ohjelmistopaketti, joka pitää sisällään kaiken käännetyn datan yksityiskohtaisesti yhdessä tiedostossa.

Julkaisu tehdään joko sisäisesti tai ulkoisesti. Sisäinen julkaisu on vain yhtiön omaan käyttöön ja ulkoinen julkaisu joko asiakkaalle tai maailmalle. Sisäinen julkaisu tehdään usein sähköisesti ja tiedostot helposti kaikkien saataville. Ulkoisessa julkaisussa ohjelmistopaketti toimitetaan joko verkon välityksellä tai kiinteässä muodossa, kuten CD- ja DVD -levyillä.

5 OHJELMISTOPAKETIN RAKENTAMISEN AUTOMATISOINTI

Ohjelmistopaketin kokoaminen, testaaminen ja toimitus manuaalisesti ovat erittäin aikaa kuluttavia ja virhealttiita toimenpiteitä. Tämän vuoksi kaikkea rutiininomaista ja inhimillisen erehdyksen alaista toimintaa pyritään automatisoimaan mahdollisimman pitkälle.

Automatisointiin on muutamia varteenotettavia käyttömahdollisuuksia. Ylivoimaisesti suosituin ja käytetyin keino ohjelmistoteollisuudessa ovat skriptit, jotka komentorivipohjaisesti tekevät monia toimintoja peräkkäin. Toinen hyvä automatisointikeino on käyttää valmiita malleja (engl. template), jotka toimivat kuin muotteina automatisoinnissa.

Ohjelmistopakettien automatisoinnissa on kuitenkin omat haittapuolensa, joten niiden käyttöä tulee seurata tarkasti. Pahimmassa tapauksessa automatisointi skripti voi tuhota tai sotkea käännettävän ympäristön täysin, joten varmuuskopiointi on lähes välttämätöntä ennen automatisoinnin aloittamista. Myös mallien käyttäminen ohjelmistopaketin rakennuksen automatisoinnissa sisältää muutamia virhealttiita toimenpiteitä.

```
@echo off
setlocal

goto :start
:usage
echo.
echo.*********************************************************************************
echo.* This cmd cleans the given area for folders beginning with one or
echo.* more undorscore characters [_]. In prior to use mark the folders
echo.*********************************************************************************
goto :eof
:start

if [%1] == [] @(
    goto :usage
)

if /I [%2] == [-x] @(
    SET EXCMD=rmdir /s /q
    SET EXMSG=# Removing
) ELSE (
    SET EXMSG=# -x to remove
)
CALL ECHO.### %~n0 Started: %DATE% %TIME%

IF EXIST %~f1 call :s_DoClean %~f1

CALL ECHO.### %~n0 Finished: %DATE% %TIME%
endlocal
goto :eof

:s_DoClean
pushd %1
IF NOT DEFINED EXCMD echo # Folders which would be removed if -x is defined:
FOR /D %%G IN (_*) DO (
    echo %EXMSG% %%G
    IF DEFINED EXCMD %EXCMD% %%G
)
popd
goto :eof
```

Listaus 1 Esimerkki-skripti käännösalueen puhdistamiseen

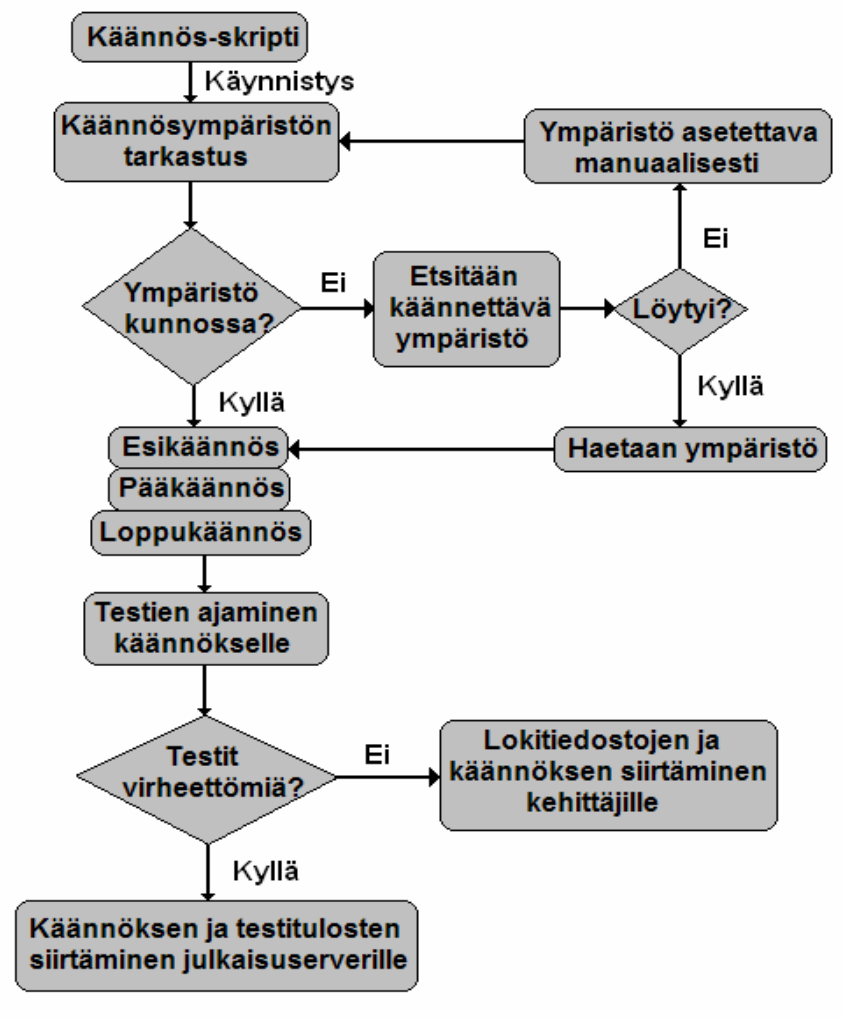
Yllä olevassa listauksessa 1 on Windows – käyttöjärjestelmälle tehty skripti, jolla voidaan puhdistaa tietyn kansion sisältä kaikki alaviivalla ’_’ alkavat hakemistot. Skriptiä tutkimalla voidaan havaita sen käyttävän erilaisia muuttujia ja kutsuja, joiden avulla haluttu kansio saa ’rmdir /s /q’ komennon, joka tuhoaa kaiken kansion sisällön. Tällainen skripti on hyödyllinen tilanteissa, joissa tiettyjen kansioiden poistot halutaan taustalle, eli hiljaiseksi toiminnaksi.

5.1 Skriptien käyttö ohjelmistojen kokoamisen automatisoinnissa

Automatisoinnissa yhdistetään monia toimintoja toimimaan peräkkäin tai yhtäaikaisesti. Peräkkäin kulkevat toiminnot suoritetaan loogisessa järjestyksessä alusta loppuun ja lisäksi ajonaikaisten vaiheiden aikana voidaan yhdistää monta toimintoa toimimaan yhtäaikaisesti. Näin pystytään optimoimaan käytössä olevat resurssit. Peräkkäin kulkevia toimintoja voivat olla esimerkiksi esikäännös, pääkäännös ja loppukäännös. Jokaisen vaiheen aikana voidaan siis kääntää montaa eri komponenttia yhtäaikaisesti. Automatisoinnissa tällaista toimintatapaa kutsutaan ketjutukseksi.

Skriptit ovat nopeita ja toimiessaan estävät inhimillisten erehtymisten mahdollisuutta. Skriptejä voidaan yleensä myös muokata omanlaisiksi, eli kustomoiduiksi. Näin saadaan erittäin monipuolisesti muunnettavia samantoimintaisia käännöksiä aikaiseksi. Kustomointi luo skripteille uusiokäyttöarvoa, joka helpottaa skriptien ylläpitäjien työtä.

Automatisoivien skriptien haittoina pidetään niiden epävakaata toimivuutta virhetilanteissa. Skriptien käyttäjien on myös tiedettävä mitä skriptit tekevät, jotta virhetilanteissa voidaan palata kohtaan, jossa virhe syntyi. Sivun 26 kuvassa 8 kuvataan käännöksen hoitavan skriptin toiminnallisuutta tilakaaviolla. Esimerkki-skriptin käynnistys suorittaa joukon erillisiä komentoja tutkimaan käännösympäristöä. Mikäli työaluetta ei löydy tai se ei ole kunnossa, skripti pyyhkii mahdollisen vanhan ympäristön tyhjäksi ja hakee uuden käännösympäristön työkansioon. Mikäli tämä onnistuu, ajetaan ympäristössä käännös ja tehdään samanaikaisesti lokitiedostoja käännöksestä. Ajon päätyttyä tutkitaan lokitiedot lävitse ja mikäli niistä ei löydy virheitä, siirretään käännös ja testitulokset julkaisupalvelimelle (engl. publishing server). Tähän voidaan lisätä myös automaattinen sähköpostin lähettäminen niille, keitä kyseinen julkaisu kiinnostaa.



Kuva 8 Käännös-skriptin yleinen tilakaavio

5.2 Mallien käyttö ohjelmistojen kokoamisen automatisoinnissa

Ohjelmistopakettien toimittaminen tehdään yleensä samoin periaattein. Ensimmäiseksi haetaan lähdekoodit käännösympäristöön, sitten käännetään valmis paketti, testataan se ja viimeisenä toimenpiteenä toimitetaan paketti eteenpäin. Työtä voidaan helpottaa, eli automatisoida, tekemällä malli (engl. template) ko. toimintasarjalle.

Malli sisältää tiedot kaikista rutiininomaisista toiminnoista, kuten käännöstyökaluista, testaustyökaluista ja lopullisesta toimitussijainnista. Käyttäjän ei tällöin tarvitse kuin asettaa malliin kaikki spesifistisen tuotteen tiedot. Näitä tietoja ovat yleensä erilaiset kolmannen osapuolen komponentit, datatiedot sekä lähdekoodin sijaintipaikat.

Mallien käytössä on olemassa omat vaaransa. Mallit ovat monesti tehty erittäin joustaviksi, jolloin virhetilanteita ei pitäisi syntyä. Tämä joustavuus saattaa joskus vaikuttaa johonkin asiaan, jota lähdekoodin kehittäjät eivät ole tarkoituksellisesti tarkoittaneet. Malleja kannattaakin käyttää vai tilanteissa, joissa projektit ovat erittäin samankaltaisia.

Malleja käytetään yleisesti nopeasti päivittyvillä web-sivuilla, joissa on käytössä tietokanta ja lähdekoodit. Malliin asetetut tiedot luovat tällöin web-sivua lukevalle käyttäjälle oikeanlaisen sivujärjestyksen. Web-sivun ylläpitäjän pystyy myös valitsemaan mallin, jonka mukaan hän pystyy lisäämään sivuille uutta tietoa ja malli tekee itse työn sivuille.

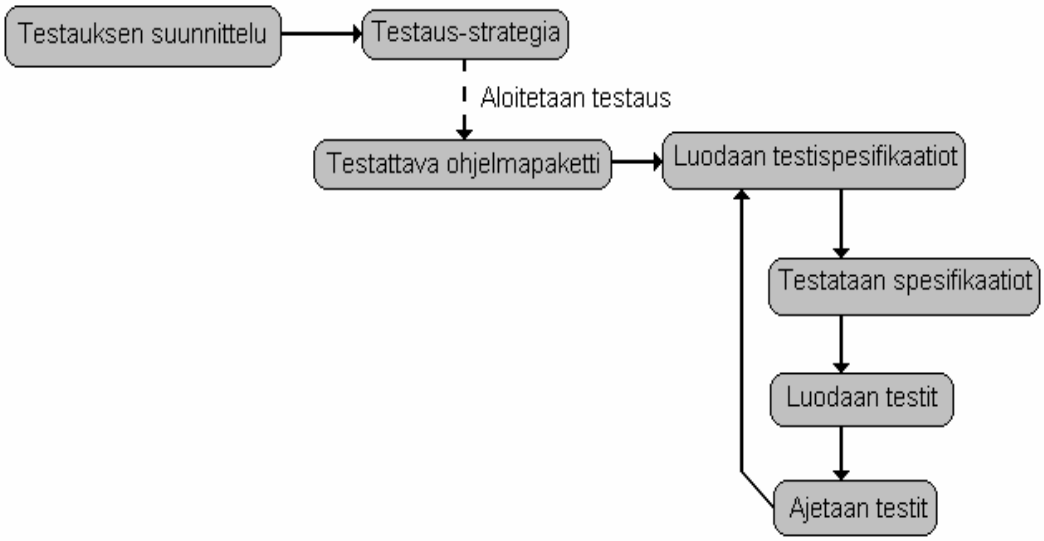
5.3 Automatisointi osana testausprosessia

Ohjelmistoprojektissa ei ole välttämätöntä tehdä testauksia manuaalisesti. Monesti automatisointi lisää testausvarmuutta, kun ohjelmisto testataan monilla erilaisilla testisyötteillä ja toiminnoilla.

Kun ohjelmistoa testataan automatisoinnilla, pitää ohjelmiston lähdekoodi ja valmiiksi käännetty ohjelma syöttää testiohjelmalle. Tällöin ohjelma lukee, sekä tulkitsee lähdekoodia ja seuraa valmista käännettyä ohjelmistoa rivi riviltä kaikki sille asetetut testitapaukset läpi. Tämäntapainen testaus vie oman aikansa, mutta säästää työvoimaa muihin työtehtäviin.

Testausprosessissa on pystyttävä hyödyntämään käänteistekniikkaa (engl. reverse engineering). Tällöin kaikki testivaiheet on tiedettävä jo valmiiksi, jotta mahdollisten odottamattomien virheiden ilmeneminen pystytään jäljittämään. Odottamattoman virheen ilmentyessä tarkastetaan automatisoinnin rakentamat lokitiedostot, jotta voidaan jäljittää millaisia testejä automatisointi oli ohjelmalle tehnyt virheen ilmetessä. Tämän jälkeen raportoidaan virhe ja käytetään virheenhallintaa hyväksi virheen korjauksessa.

Automatisoinnilla pystytään tekemään mm. sauhu-, käynnistys- ja läpiajotestit. Vaikka testitulokset näyttäisivätkin automatisoinnin jälkeen olevan kunnossa, ei koneen omatekemiin testauksiin saada kuitenkaan aina täyttä varmuutta.



Kuva 9 Automaattisen testauksen toimintamalli

Kuvan 9 mukaisesti, testauksen suunnitelmasta luodaan strategia, joka sisältää tiedot siitä, missä järjestyksessä testit on ajettava läpi. Kun looginen testirunko on tehty, aloitetaan testattavan ohjelmapaketin testaaminen. Tämän jälkeen siirrytään luomaan testispesifikaatiot, jotka sisältävät mm. raja-arvot muuttujille, joilla testit ajetaan läpi. Kun nämä rajat on annettu ja todettu toimiviksi, tehdään itse testit, jotka ajetaan ohjelman lävitse. Testien tuloksia hyväksikäyttäen siirrytään takaisin

luomaan testispesifikaatiot uusista testattavista arvoista. Kun kaikki testaustulokset on käyty lävitse, asetetaan ohjelmapaketin tila toimitusvalmiiksi.

6 YHTEENVETO

Ohjelmistopakettien kokoamisessa on noudatettava sääntöjä, jotta voidaan ennaltaehkäistä monta ongelmaa. Säännöt asetetaan konfiguraationhallinnan avulla, joka sisältää tiedot siitä, miten projekti etenee eri vaiheissa. Ohjelmistopakettien kokoaja suorittaa ohjelmistopakettien kokoamisen hallitusti näiden sääntöjen avulla ja samalla hän pyrkii saamaan toistuvista mekaanisista töistä mahdollisimman automatisoituja. Automatisoinnissa voidaan käyttää apuna joko malleja tai skriptejä. Näistä kahdesta skriptit ovat selvästi yleisempiä ja niitä on myös helpompi käyttää ja rakentaa.

Ohjelmistopakettien kokoaja käyttää työssään pääasiallisesti versionhallintatyökalua. Tämä työkalu hoitaa tiedostojen siirrot kannan ja työalueen välillä. Ohjelmistopakettien kokoamisessa työ hoidetaan yleensä komentorivipohjaisesti. Tällöin on mahdollista käyttää erilaisia skriptejä tekemään mekaaninen työ ja samalla pystytään ennaltaehkäisemään inhimillisten virheiden syntyminen, kuten kirjoitusvirheet. Mikäli virheitä ilmenee ohjelmistopakettien käännösvaiheessa, noudatetaan virheenhallintaa virheen korjaamiseksi. Virheet luokitellaan yleensä kriittisyyden mukaan.

Käännösympäristön kokoaminen hoidetaan suurissa projekteissa etäyhteytenä nopealle käännöskoneelle. Ympäristö tarvitsee käännettävän materiaalin ja päivitetyt työkalut, jotta käännös voidaan toteuttaa. Käännös saattaa usein rakentua monivaiheisesti.

Kun konfiguraationhallinnan sääntöjä noudatetaan, sekä ohjelmistopakettien kokoamisen työkaluja käytetään oikein, saadaan laadukas ohjelmistopaketti. Kun ohjelmistopaketin kokoaja on saanut ohjelmistopaketin rakennettua, ajetaan sille tietynlaiset testit, jotka varmistavat paketin toimivuuden. Tämän jälkeen paketti on valmis toimitettavaksi eteenpäin joko asiakkaalle, firman sisäiseen käyttöön tai maailmalle julkaistavaksi.

LÄHDELUETTELO

1. Seppänen, Vesa, Konfiguraationhallinta. Helsingin yliopisto. Tietojenkäsittelytieteen laitos. Helsinki 2000 11s. http://www.cs.helsinki.fi/u/laine/otv/seppanen.pdf

2. American National Standard, ANSI/IEEE 1042 – 1987, 03.04.2007 http://ieeexplore.ieee.org/iel1/2592/2920/00089631.pdf?isnumber=2920&prod=STD&arnumber=89631&arSt=&ared=&arAuthor=

3. Kuosmanen, Juha, Konfiguraationhallinta ja Rational ClearCase. Helsingin yliopisto. Tietojenkäsittelytieteen laitos. Helsinki 2000 12s. http://www.cs.helsinki.fi/u/laine/otv/kuosmanen.pdf

4. Telelogic, Configuration Management to improve communication and collaboration, 26.03.2007 http://www.telelogic.com/Products/synergy/synergycm/index.cfm

5. DMOZ Open Directory project, Configuration Management Tools, 26.03.2007 http://dmoz.org/Computers/Software/Configuration_Management/Tools/

6. Oy Laatukonsultointi P.Kantelinen Ab, Software Configuration Management Tools, 26.03.2007 http://www.laatuk.com/tools/SCM_tools.html